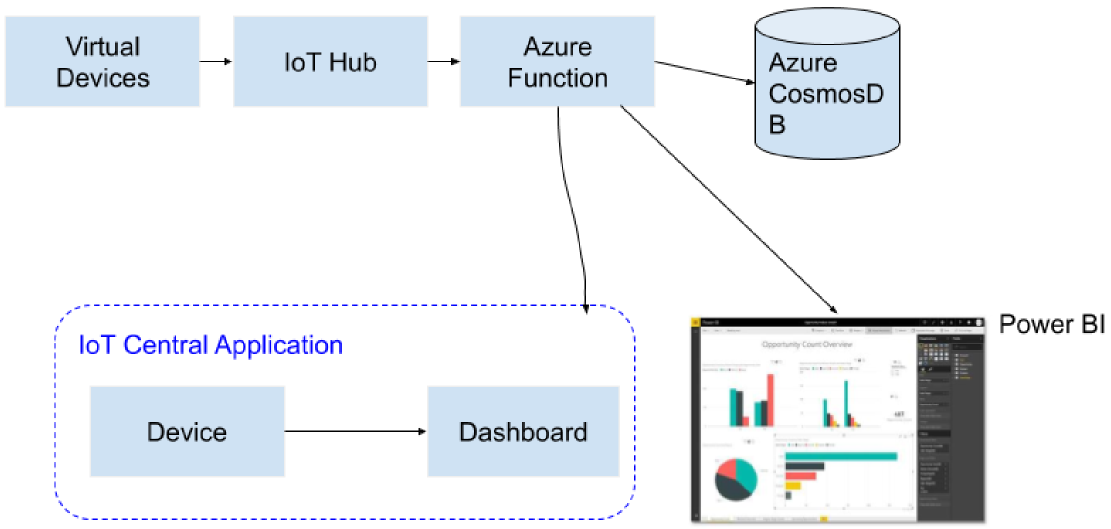
Azure IoT Central PoC Demo

This is a demo to show how Azure IoT Hub, Azure IoT Central Services and Power BI can be used to deliver an end-to-end IoT solution,from the devices to visualization in IoT Central dashboard.

Overview of The System

The system consists of a simple and common IoT flow, where there is a device (which will be simulated), it securely connects to an endpoint in the cloud and sends its telemetry. Device telemetry will then be processed and transformed in a particular manner before it is visualized for business intelligence.

The architecture below visualizes the major components of the solution:



To put the solution together, the following Cloud Services will be used:

- **Azure IoT Hub** - IoT Hub is a managed service, hosted in the cloud, that acts as a central message hub for bi-directional communication between your IoT application and the devices it manages.
- **Azure IoT Central** - Azure IoT Central is a fully managed SaaS (software-as-a-service) solution that makes it easy to connect, monitor and manage your IoT assets at scale.
- **Azure Functions** - Azure Functions is a serverless compute service that lets you run event-triggered code without having to explicitly provision or manage infrastructure.

- **Azure Cosmos DB** - Azure Cosmos DB is Microsoft’s globally distributed, multi-model database service for operational and analytics workloads. It offers multi-mastering feature by automatically scaling throughput, compute, and storage.
- **Power BI Service (Optional)** - Power BI is a business analytics solution that lets you visualize your data and share insights across your organization.

What You'll need

- An Azure account with an active subscription
- A PowerBI Pro License Account
- A text editor
- Deployment code and files which can be found in this repository
- Basic knowledge of any programming language

Instructions

Resource Provisioning

1. You'll use the Azure CLI to deploy all of the resources in the solution.The Azure command-line interface (CLI) is Microsoft's cross-platform command-line experience for managing Azure resources. Head over to https://docs.microsoft.com/en-us/cli/azure/install-azure-cli?view=azure-cli-latest and download the latest version of the Azure CLI appropriate for your operating system.

2. After installing the Azure CLI, run

```
    $ az login
```

   This will log you into the approriate azure account you want to use.

3. Start by creating a resource group. A resource group is a container that holds related resources for an Azure solution.You may replace *example* with the name you want to give to your resource group but remember to change to that name whenever we need to deploy additional resource. Depending on where you are located, you may want to change the location, but this is not important for now.

```
$az group create --name example --location "East US"
```

Take a note of the resource group name you used above, you will need it in subsequent steps.

4. Now that you have a nice container for your resources, you can begin deploying the required resources. You will deploy the Azure IoT Hub, a Function App (where all your Azure Functions will be hosted) and the Azure Cosmos DB for data storage. The deployment template and a parameter file is provided in this repository. cd into the *deployment_templates* folder. There are two files, the template.json files describes the resources that will be provisioned in Azure and the *parameters.json* are the parameters that will be used to deploy these resources.

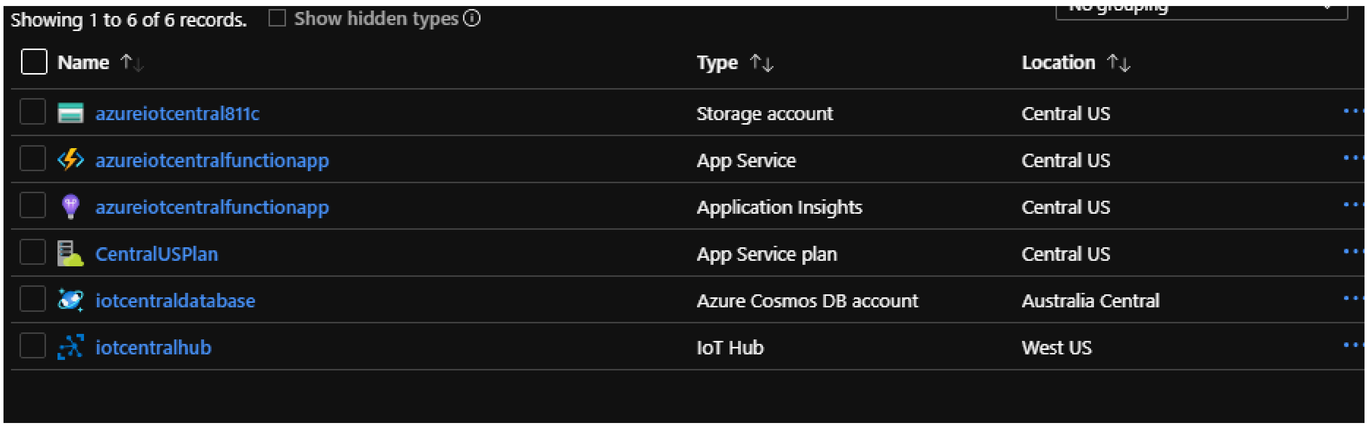
5. Open the *parameters.json* file in your text editor. On a property called *parameters* you will find all the necessary resources that the template needs to deploy the resources. They all have a value of `null`. For each, input a unique string(preface it with your name or the name of your company e.g mycompanymainDB to ensure the names are globally unique). Avoid special charcaters as some of the resources accept only letters.

6. Run the following command, changing the `example` resource group name to whatever you noted down. The deployment will take some time. Wait for it to finish, in the mean time, you can check your resource group in the portal and confirm resources are being deployed in it.

```
$ az group deployment create --name ExampleDeployment --resource-group
example --template-file template.json --parameters parameters.json
```

The most likely issue you might encounter when deploying the resources is errors to do with unique or illegal values in your parameters file. Please note that the CLI will deploy the other resources with valid names even if one or more of the resources fail to deploy as a result of invalid names entered. After deployment, if you get an error message, read the error, and change only the parameter value causing the error and run the above command again. The resources will not be duplicated if they are already deployed.

7. If you view your resource group on Azure Portal, you should see 6 resources as shown below:



Showing 1 to 6 of 6 records. Show hidden types

Name	Type	Location
azureiotcentral811c	Storage account	Central US
azureiotcentralfunctionapp	App Service	Central US
azureiotcentralfunctionapp	Application Insights	Central US
CentralUSPlan	App Service plan	Central US
iotcentraldatabase	Azure Cosmos DB account	Australia Central
iotcentralhub	IoT Hub	West US

The first 4 resources (Storage account,App Service, Application Insights, App Service Plan) are resources to support your functions, the Azure Cosmos DB account is where all your databases will live and the IoT Hub is your cloud endpoint for devices to connect to.

8. Add the Azure IoT CLI extension to the Azure CLI by running:

```
$az extension add --name azure-cli-iot-ext
```

You will need this later.

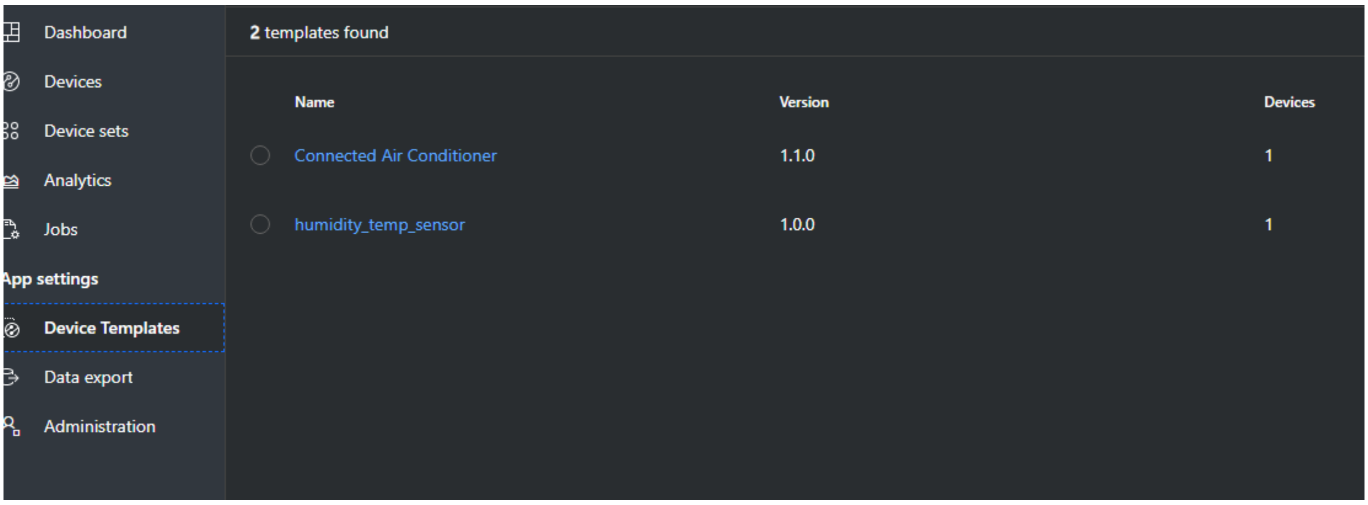
9. Next, deploy the Azure Central Application that will receive and visualize the processed IoT telemetry. Run the following command, changing the `--resource-group` parameter (to the one you noted down), the `--name` and the `--subdomain`. You can change the `--display-name` to have a custom display name on the Azure Central site. The `--subdomain` and `--name` parameter must be unique, so

come up with a random string (use your name or the name of your company plus some random characters). You may need to change these parameters a few times before the deployment succeeds.

```
$az iotcentral app create --resource-group "example" --name
"myiotcentralapp67ramds" --subdomain "myuniquesubdomain" --sku S1 --template
"b922fba8-b44c-46e9-8e1f-c44b95bac98a" --display-name
"HumidityandTempSensor"
```

10. Log back to the portal and confirm the Azure IoT Central Application was deployed. Click on the resource. There isn't much you can do with the Application from the Portal, but from there you will get the application's url. Follow the URL and you will land on the Dashboard of the IoT Central Application.

11. On the Left Side bar, under **App settings** click on **Device Templates**.

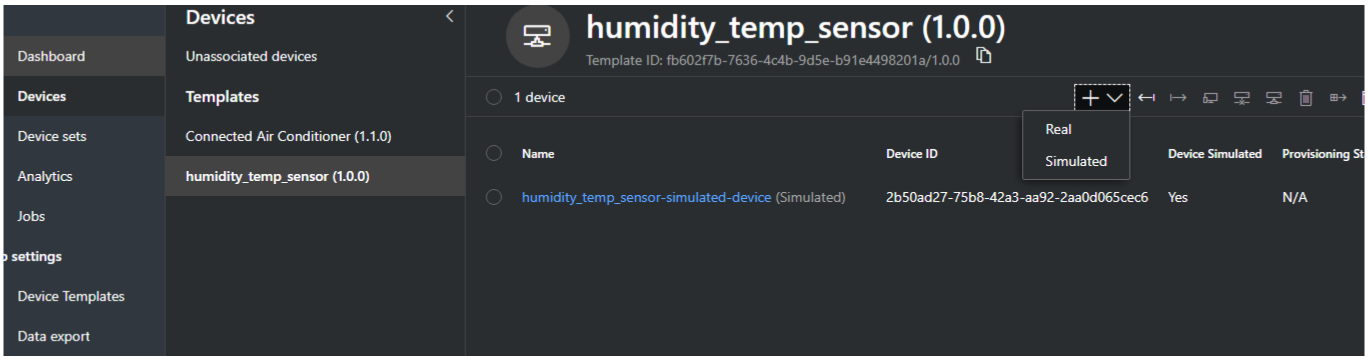


2 templates found

Name	Version	Devices
Connected Air Conditioner	1.1.0	1
humidity_temp_sensor	1.0.0	1

Click the **humidity_temp_sensor** template. You will see the configured measurements that the template expects from the device. The visuals will start to populate with some values, these are simulated values from the app itself, it just gives you an idea of how everything will be working.

12. The **Device Template** is only a blue print of what data points the device is sending, you still need to configure an actual device. On the side navigation, click on **Devices**. You should see the the **humidity_temp_sensor** template. Click on it. Notice there is already a device for this template, that was generated automatically and it simulates your device data. Select **+**, then Real. Click **Create**



humidity_temp_sensor (1.0.0)

Template ID: fb602f7b-7636-4c4b-9d5e-b91e4498201a/1.0.0

1 device

Name	Device ID	Device Simulated	Provisioning St
humidity_temp_sensor-simulated-device (Simulated)	2b50ad27-75b8-42a3-aa92-2aa0d065cec6	Yes	N/A

The device is now created and ready to receive telemetry.

Device Registration and Simulation

13. All the required resources are set up. Referring back to the system architecture, the first block in the flow is the virtual device (or a real one if you have one). Before a device is allowed to talk to the IoT Hub, it needs to be registered with that particular Azure IoT Hub. Run the command below on the terminal to register and create a device in the IoT Hub. You can log back into the portal and confirm the name of your IoT hub, change the `--device-id` to your preferred name.

```
$ az iot hub device-identity create --hub-name {YourIoTHubName} --device-id
mySimulatedDevice
```

14. The device is registered but you need its connection string which has the endpoint and keys. To get this, run the following command.

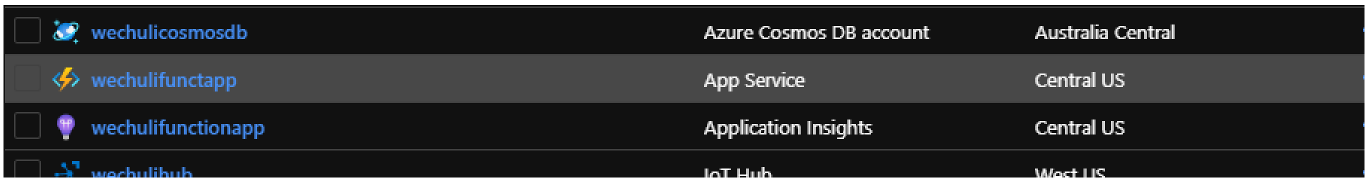
```
$ az iot hub device-identity show-connection-string --hub-name
{YourIoTHubName} --device-id mySimulatedDevice
```

Copy the returned `connectionString` value and store it somewhere, you will need it in the preceeding steps.

15. You will also need to create a consumer group which will be used by Azure Functions to pull data from the IoT Hub. Run the following command replacing the appropriate name with the name you called you're IoT Hub. You can leave the name of the consumer group to `data` but if you choose to change it, remember to also change it in subsequent steps that require the name of the consumer group.

```
$az iot hub consumer-group create --hub-name {YourIoTHubName} --name data
```

16. Next is to make the virtual device, for convenience purposes, a function will be used to simulate data sent from a real device. You could run this on your own computer or on an actual device. JavaScript (NodeJS) will be used to write the simple application to simulate the device. Head back to the portal and click on the on the Function App (it is listed as an App Service resource type)



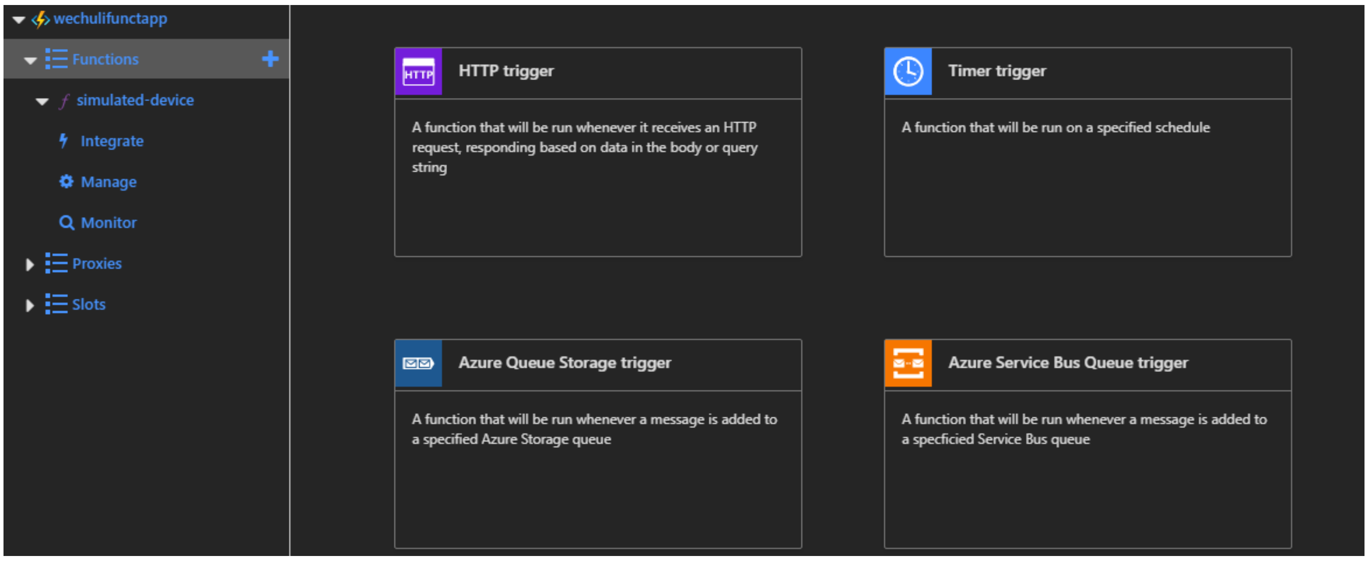
wechulicosmosdb	Azure Cosmos DB account	Australia Central
wechulifunctapp	App Service	Central US
wechulifunctionapp	Application Insights	Central US
wechulihub	IoT Hub	West US

16. You will need to configure the Function application with some configuration settings before you start writing the functions.

- Go back to your resource group and select the **Storage account** resource. Under **Settings**, click on **Access keys**. Copy the Connection string of **key1**.
- Click on the Function App again. At the top, click on **Platform features**, then select **Configuration** which is under **General Settings**.
- Under **Application settings**, Click on the **+** to add **New application setting**. The Name of the setting is **AzureWebJobsStorage**, and the Value is the connection string you copied from the Azure Storage account. Click **OK**.

- As you did above, add additional application settings with the below parameters:
    - **Name**: WEBSITE_NODE_DEFAULT_VERSION **Value**: 10.14.1
    - **Name**: FUNCTIONS_EXTENSION_VERSION **Value**: ~2
    - **Name**: FUNCTIONS_WORKER_RUNTIME **Value**: node
- Save the settings.

17. The first function you will create will simulate the virtual device. A convenient way to do this is to use a timer function that will be triggered periodically to send data to the IoT Hub (the same way a real device would). Click on the **+** under your Function App to create a function. Click on **Timer trigger**. If you don't see this right away, select **In-portal** -> **Continue** -> Select **More templates...** -> Click on **Finish and view templates**.



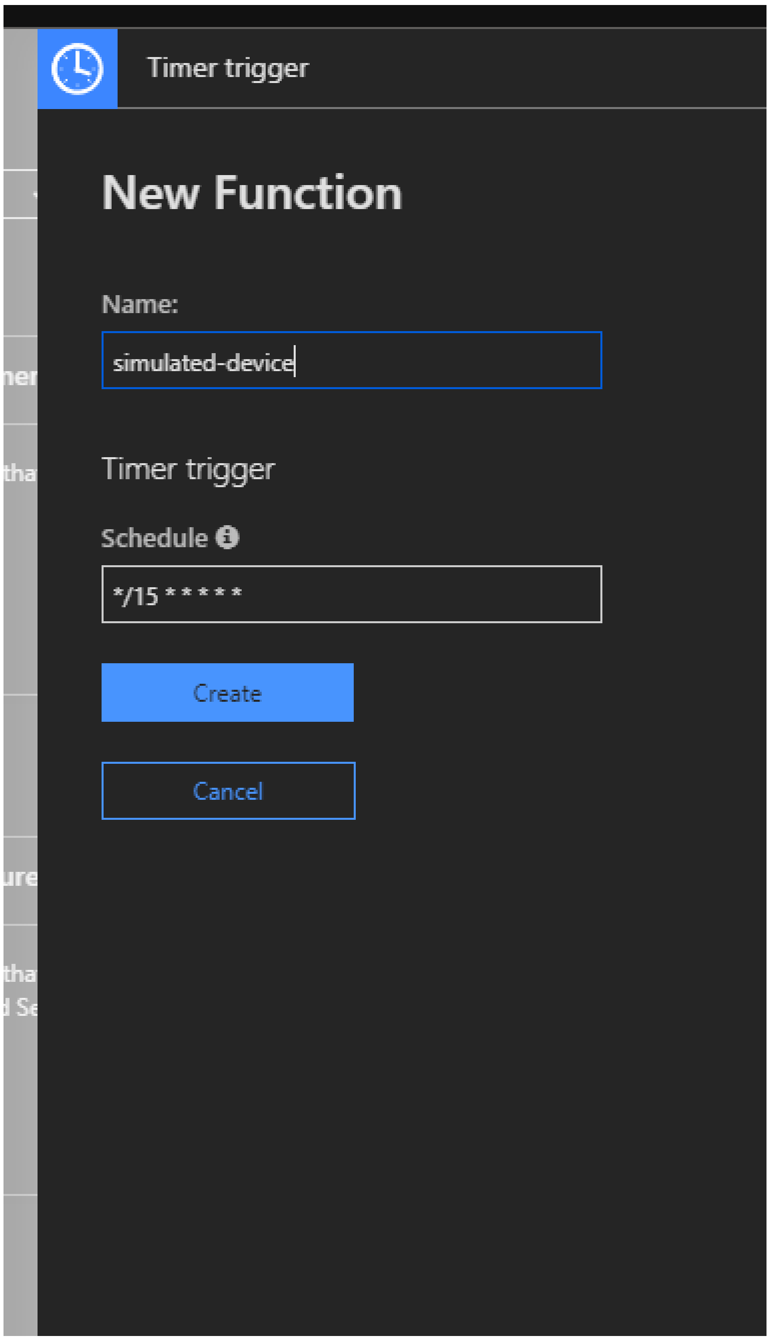
A function that will be run whenever it receives an HTTP request, responding based on data in the body or query string

A function that will be run on a specified schedule

A function that will be run whenever a message is added to a specified Azure Storage queue

A function that will be run whenever a message is added to a specficied Service Bus queue

Give the function a descriptive name, like **simulated-device**. On the Schedule, input `*/15 * * * * *`, this will fire the function every 15 seconds. Click create.



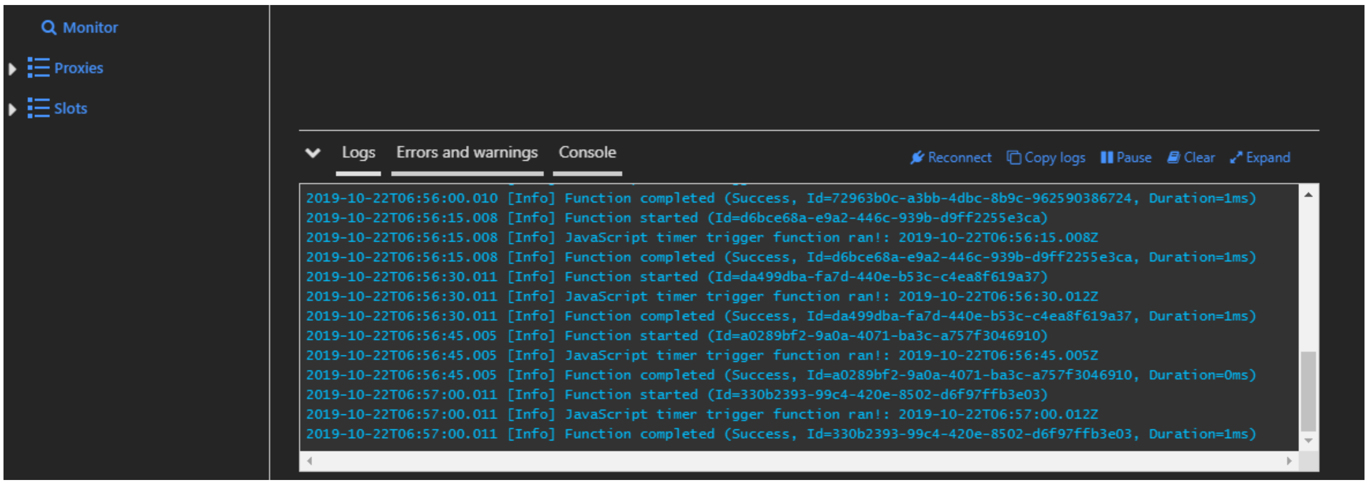
New Function

Name: simulated-device

Timer trigger

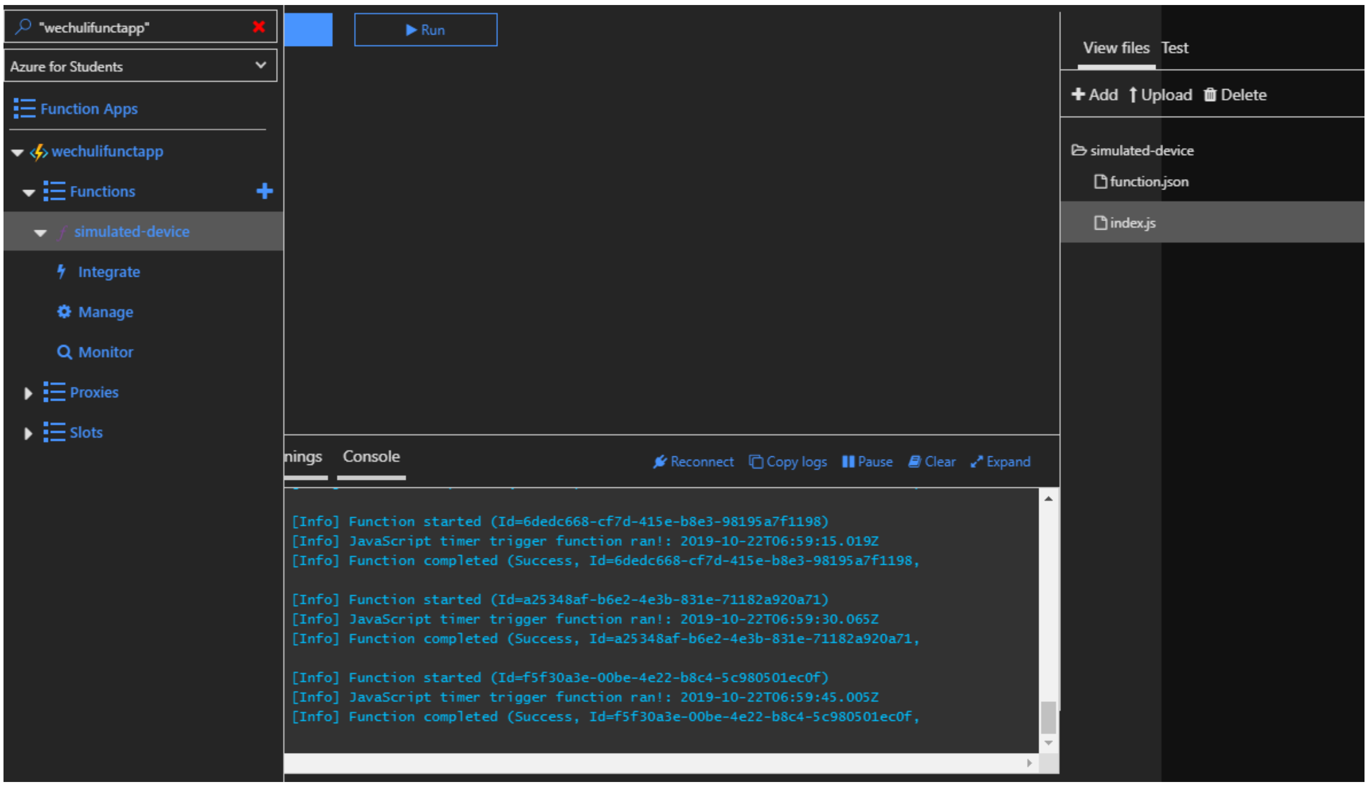
Schedule: */15 * * * * *

To confirm your function is actually running, click on it, this should open the index.js file of the function, at the bottom of the pane, click on the **Logs** tab to expand it. Your output should be similar to the below illustrations (the logs may have a slight delay).



```
2019-10-22T06:56:00.010 [Info] Function completed (Success, Id=72963b0c-a3bb-4dbc-8b9c-962590386724, Duration=1ms)
2019-10-22T06:56:15.008 [Info] Function started (Id=d6bce68a-e9a2-446c-939b-d9ff2255e3ca)
2019-10-22T06:56:15.008 [Info] JavaScript timer trigger function ran!: 2019-10-22T06:56:15.008Z
2019-10-22T06:56:15.008 [Info] Function completed (Success, Id=d6bce68a-e9a2-446c-939b-d9ff2255e3ca, Duration=1ms)
2019-10-22T06:56:30.011 [Info] Function started (Id=da499dba-fa7d-440e-b53c-c4ea8f619a37)
2019-10-22T06:56:30.011 [Info] JavaScript timer trigger function ran!: 2019-10-22T06:56:30.012Z
2019-10-22T06:56:30.011 [Info] Function completed (Success, Id=da499dba-fa7d-440e-b53c-c4ea8f619a37, Duration=1ms)
2019-10-22T06:56:45.005 [Info] Function started (Id=a0289bf2-9a0a-4071-ba3c-a757f3046910)
2019-10-22T06:56:45.005 [Info] JavaScript timer trigger function ran!: 2019-10-22T06:56:45.005Z
2019-10-22T06:56:45.005 [Info] Function completed (Success, Id=a0289bf2-9a0a-4071-ba3c-a757f3046910, Duration=0ms)
2019-10-22T06:57:00.011 [Info] Function started (Id=330b2393-99c4-420e-8502-d6f97ffb3e03)
2019-10-22T06:57:00.011 [Info] JavaScript timer trigger function ran!: 2019-10-22T06:57:00.012Z
2019-10-22T06:57:00.011 [Info] Function completed (Success, Id=330b2393-99c4-420e-8502-d6f97ffb3e03, Duration=1ms)
```

18. Click on the **simulated-device** function, on your far right, you should see the files that the function needs.Currently, you should have two files; **function.json** and **index.js**.



```
[Info] Function started (Id=6dedc668-cf7d-415e-b8e3-98195a7f1198)
[Info] JavaScript timer trigger function ran!: 2019-10-22T06:59:15.019Z
[Info] Function completed (Success, Id=6dedc668-cf7d-415e-b8e3-98195a7f1198,

[Info] Function started (Id=a25348af-b6e2-4e3b-831e-71182a920a71)
[Info] JavaScript timer trigger function ran!: 2019-10-22T06:59:30.065Z
[Info] Function completed (Success, Id=a25348af-b6e2-4e3b-831e-71182a920a71,

[Info] Function started (Id=f5f30a3e-00be-4e22-b8c4-5c980501ec0f)
[Info] JavaScript timer trigger function ran!: 2019-10-22T06:59:45.005Z
[Info] Function completed (Success, Id=f5f30a3e-00be-4e22-b8c4-5c980501ec0f,
```

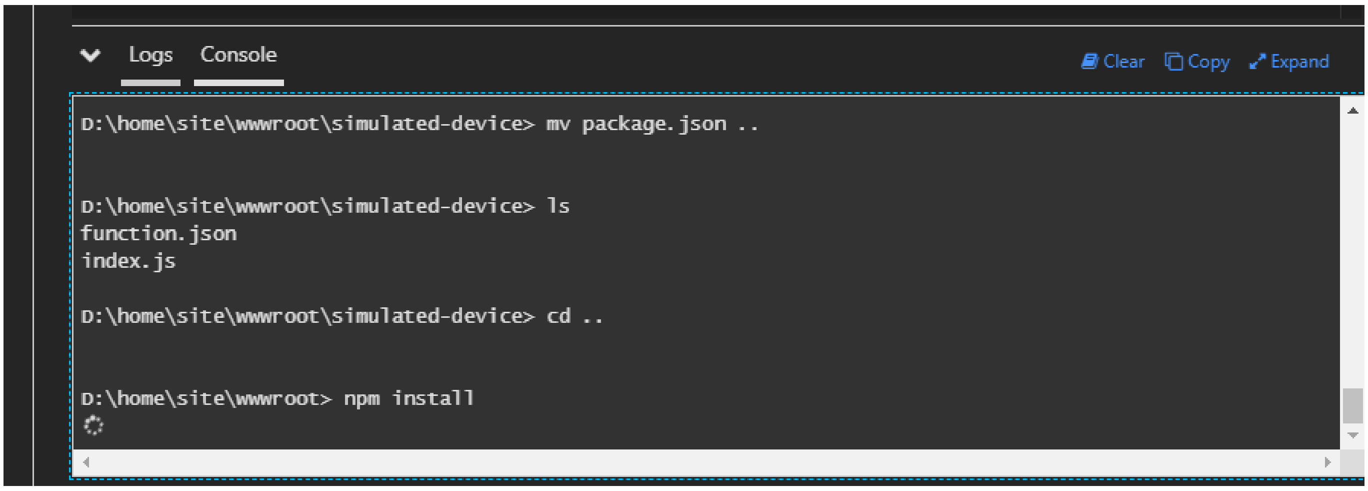
On the top level directory in this repository, there is a folder named **Function App** that contains the functions you will deploy and a `package.json` file which defines our entire Function App.

19. The **Function App** folder contains a `package.json` file. The goal is to use this file to install all dependencies that all the functions will need to run. On the Portal, under the **simulated-device** function, click on **View files** on your far right then click on **Upload**. Upload the `package.json` file in from the local directory (in the Function App folder in this directory).

20. Once the `package.json` file has been uploaded, we need to move it to the root of our Function App in the cloud (the exact directory structure we have in our local folder). So at the bottom pane, click on **Console**. You should be in the **simulated device** folder (ls to make sure.). This is where you uploaded the `package.json` file. To move that file one directory up, run the following command on the console

    ```
    $mv package.json ..
    ```

21. Go to the root (one level up) of your Function App by typing `cd ..` in the console. Confirm that the `package.json` file was moved here by typing `ls`. Finaly run `npm install` to install all the required node modules. This will take a while.



```
D:\home\site\wwwroot\simulated-device> mv package.json ..


D:\home\site\wwwroot\simulated-device> ls
function.json
index.js

D:\home\site\wwwroot\simulated-device> cd ..


D:\home\site\wwwroot> npm install
```

22. On your computer, open the **simulated-device** folder under **Function App** (Function App/simulated-device) in this repository. With a text editor such as VSCode, open the `index.js` file that is inside. This code sends simulated temperature and humidity values(and the device Id) to an IoT Hub endpoint. You need to edit the connection string value of the IoT Hub to send it to your configured hub and device (the IoT Hub connection string you noted down while registering the device).

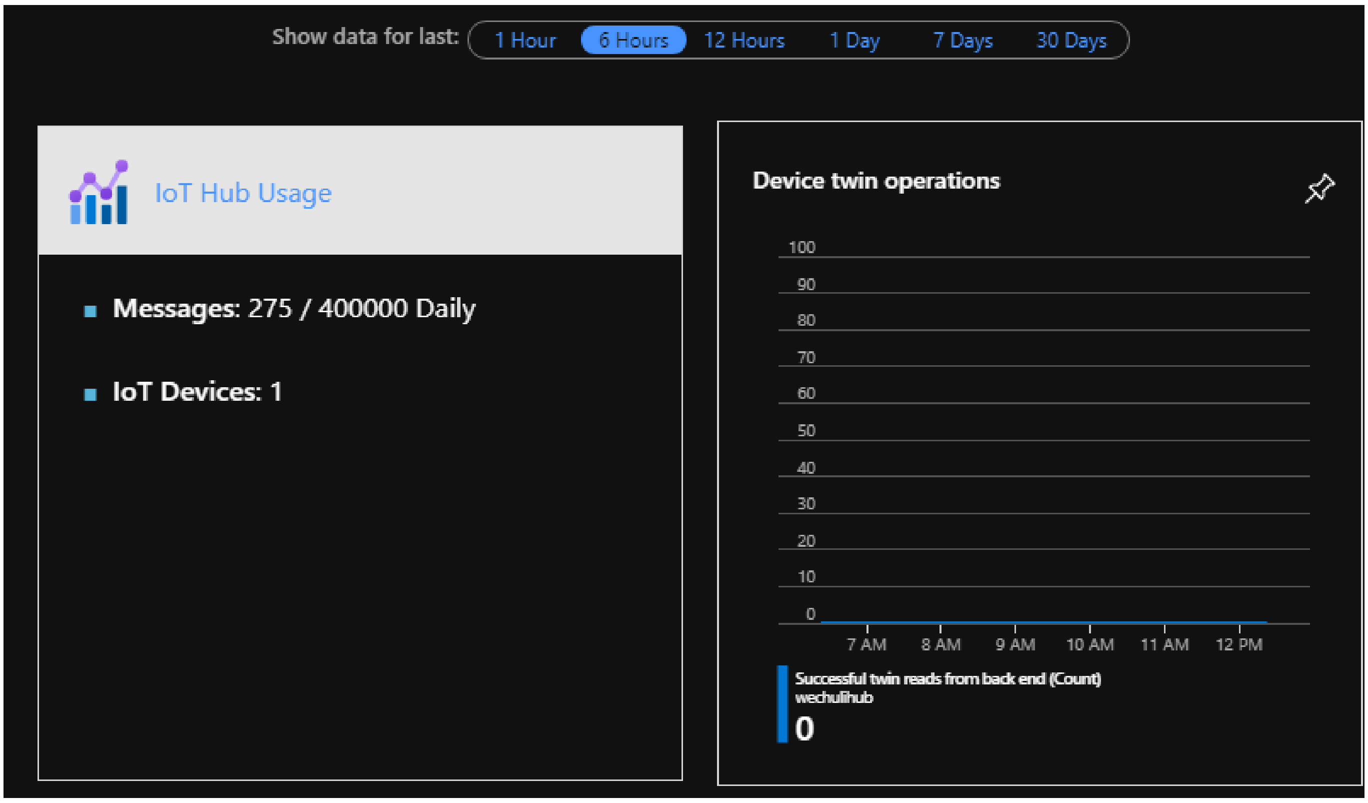
```
// Using the Azure CLI:
// az iot hub device-identity show-connection-string --hub-name {YourIoTHubName} --device-id MyNodeDevice --output table
var connectionString =
  "{input your connection string here}";
```

Save the file.

23. Back in the Azure Portal and in the Function App. Click on the **simulated-device** function. On the tab on the far right, click **View files**, then **upload**. Upload the local **index.js** (the one you added your device connection string) from **simulated-device** folder in this repository.

24. Restart the Function App. There might be some delay before you start seeing anything on the Logs.

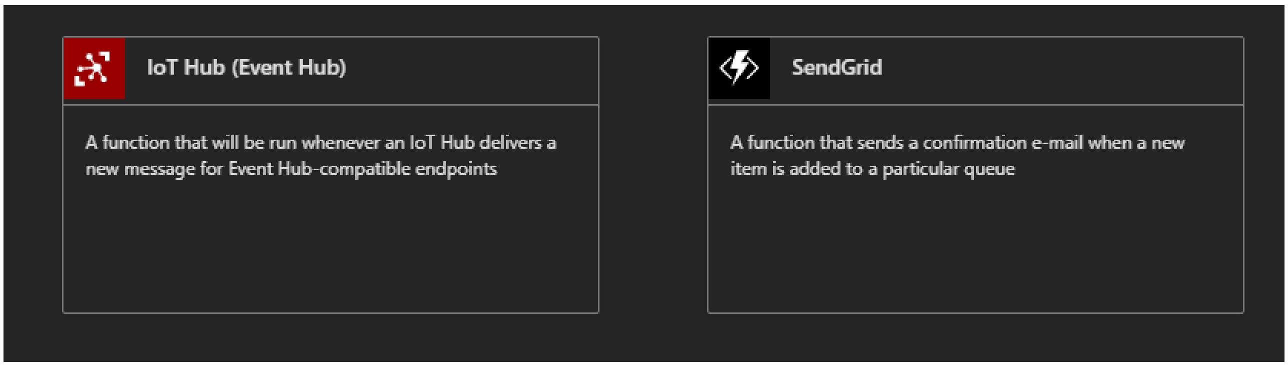
25. The Function should be sending telemetry to your IoT Hub every 15 seconds. Go back to your resource group and click on the IoT Hub. Under, the **Overview** Tab, scroll to bottom, there is a Visualization card named **IoT Hub Usage**, on it you will see the number of messages sent today (if it it more than zero, then the device simulator function is working correctly).



- **Messages:** 275 / 400000 Daily
- **IoT Devices:** 1

Successful twin reads from back end (Count)
wechulihub
0

26. The messages are successfully arriving at the IoT Hub, you need to take these messages and process them before forwarding them to the Azure IoT Central application (and optionally the Power BI service). Again, Azure Functions will be used for this task, but this time instead of the trigger being a timer, the trigger will be a message arriving at the IoT Hub end point.
27. Head back to your Function App and add another function. This time, select the **IoT Hub (Event Hub)** template.



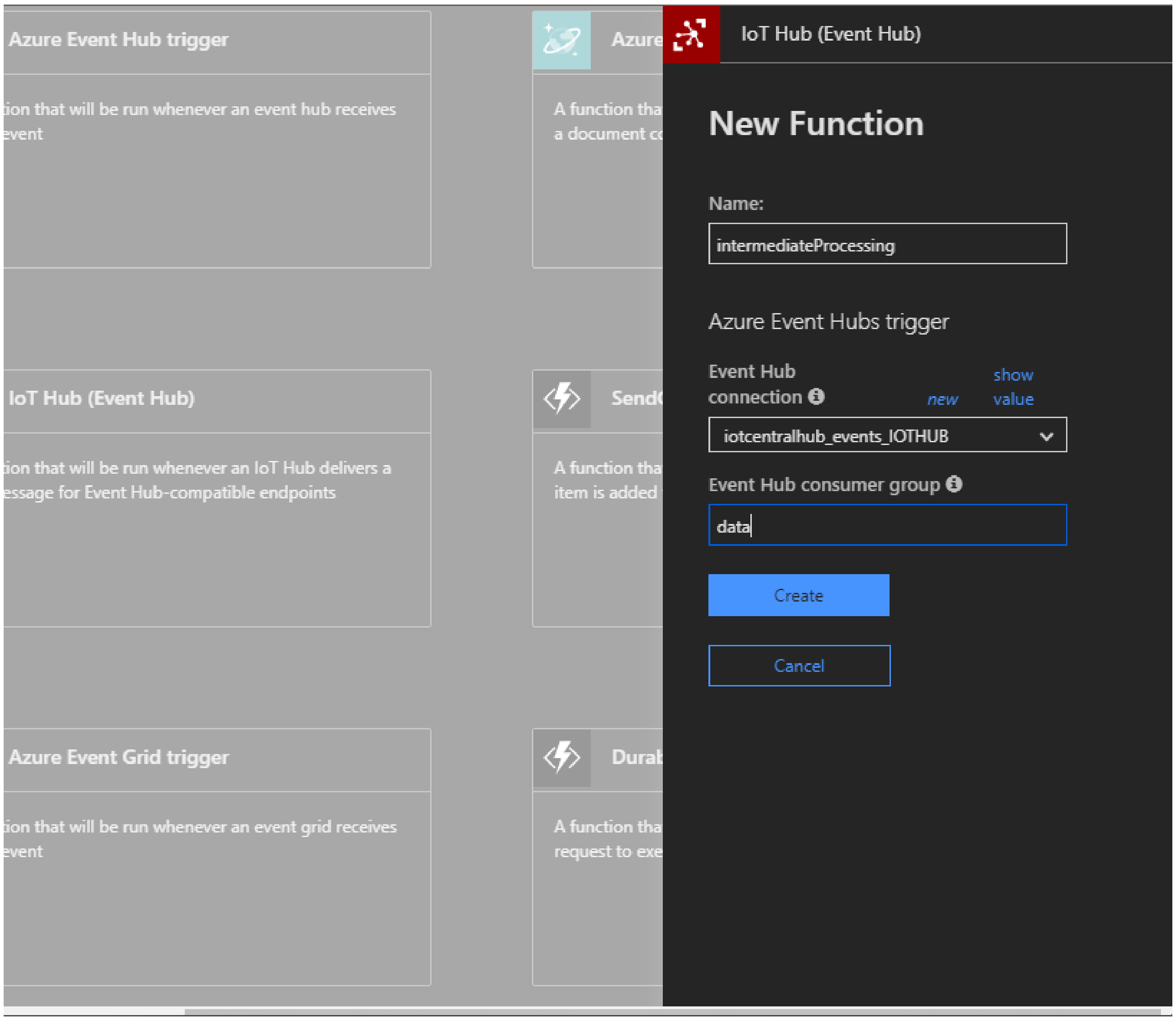
IoT Hub (Event Hub)

A function that will be run whenever an IoT Hub delivers a new message for Event Hub-compatible endpoints

SendGrid

A function that sends a confirmation e-mail when a new item is added to a particular queue

28. Install the required extensions (this may take a while). Give the Function a descriptive name like `intermediateProcessing`. Under **Event Hub connection**, click **new**, then choose **IoT hub**, on the drop down that appears, choose the appropriate IoT hub you configured for the resource group (the one you registered the device in).Under **Event Hub consumer group**, input `data` which is the name of the consumer group you created on the IoT Hub. Click **Create**


Azure Event Hub trigger
ion that will be run whenever an event hub receives event
IoT Hub (Event Hub)
ion that will be run whenever an IoT Hub delivers a essage for Event Hub-compatible endpoints
Azure Event Grid trigger
ion that will be run whenever an event grid receives event
Azure
A function tha a document co
Send
A function tha item is added
Durab
A function tha request to exe

IoT Hub (Event Hub)

New Function

Name: intermediateProcessing

Azure Event Hubs trigger

Event Hub connection: iotcentralhub_events_IOTHUB (new, show value)

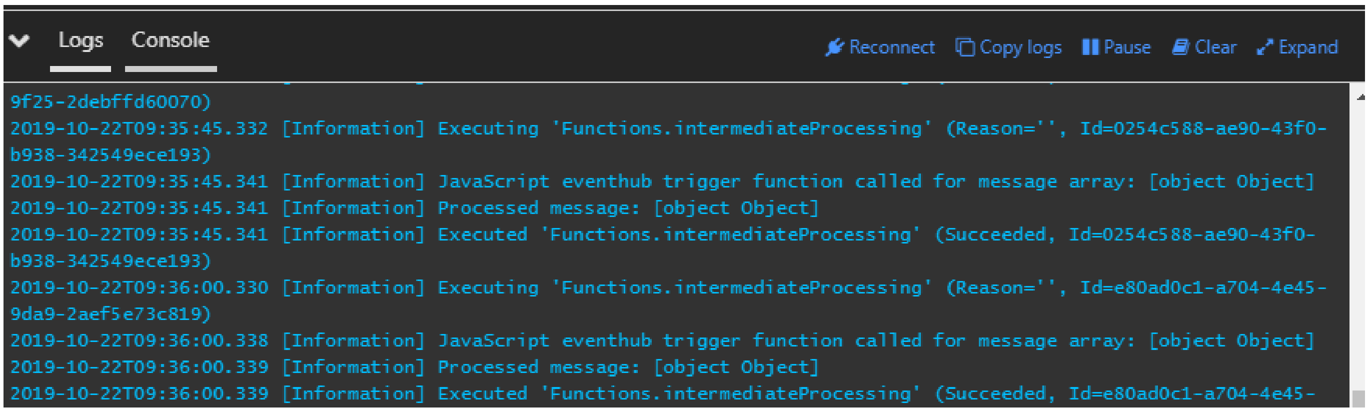
Event Hub consumer group: data

Create

Cancel

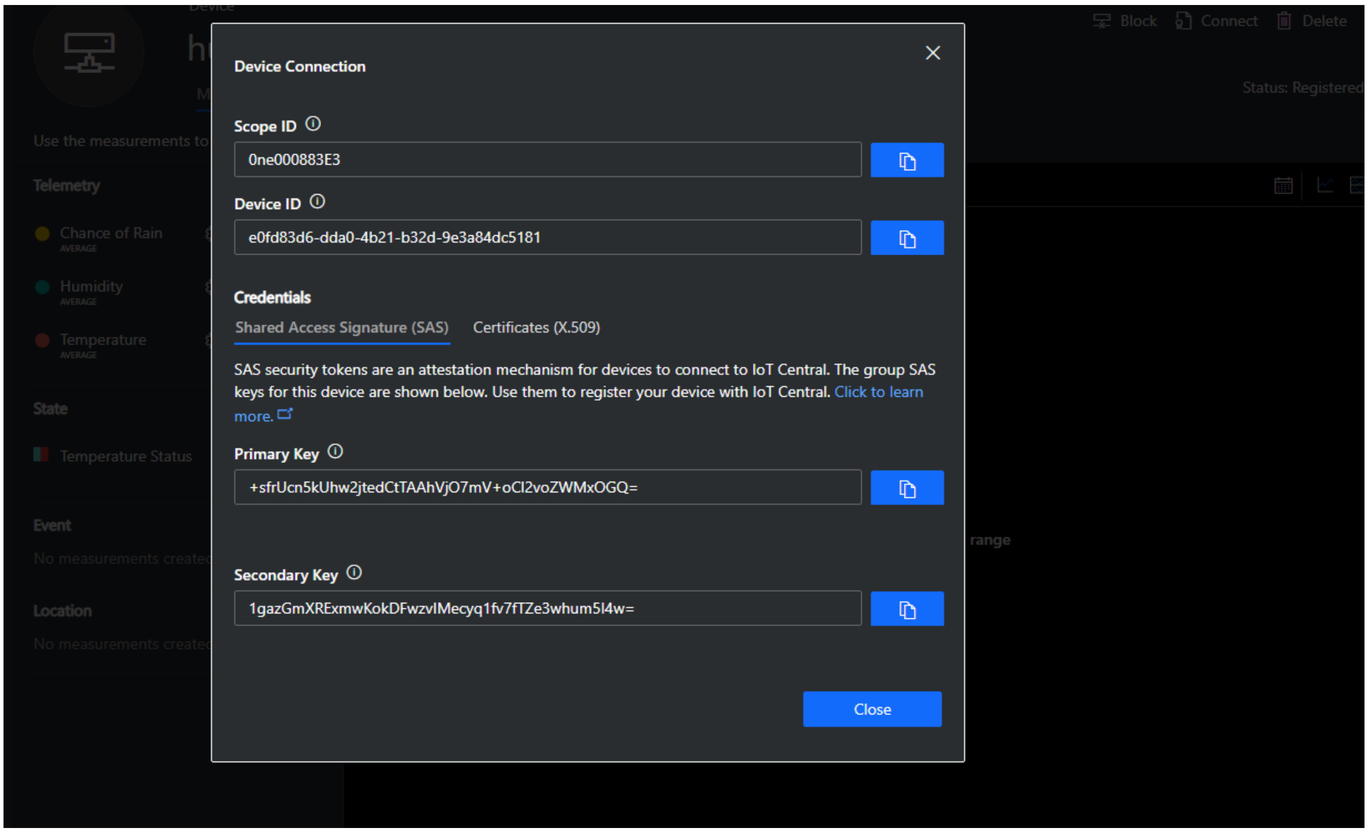
.

29. If you check the newly created function's (intermediateProcessing) logs you should get a similar output as the one shown below:



```
9f25-2debffd60070)
2019-10-22T09:35:45.332 [Information] Executing 'Functions.intermediateProcessing' (Reason='', Id=0254c588-ae90-43f0-
b938-342549ece193)
2019-10-22T09:35:45.341 [Information] JavaScript eventhub trigger function called for message array: [object Object]
2019-10-22T09:35:45.341 [Information] Processed message: [object Object]
2019-10-22T09:35:45.341 [Information] Executed 'Functions.intermediateProcessing' (Succeeded, Id=0254c588-ae90-43f0-
b938-342549ece193)
2019-10-22T09:36:00.330 [Information] Executing 'Functions.intermediateProcessing' (Reason='', Id=e80ad0c1-a704-4e45-
9da9-2aef5e73c819)
2019-10-22T09:36:00.338 [Information] JavaScript eventhub trigger function called for message array: [object Object]
2019-10-22T09:36:00.339 [Information] Processed message: [object Object]
2019-10-22T09:36:00.339 [Information] Executed 'Functions.intermediateProcessing' (Succeeded, Id=e80ad0c1-a704-4e45-
```

30. Go back to the web dashboard of the Azure Central Application you created. Click on **Devices** then choose the **humidity_temp_sensor**, select the real device you made, at the top of the device dashboard, click **Connect**.


Device Connection

Scope ID
0ne000883E3

Device ID
e0fd83d6-dda0-4b21-b32d-9e3a84dc5181

Credentials

Shared Access Signature (SAS) Certificates (X.509)

SAS security tokens are an attestation mechanism for devices to connect to IoT Central. The group SAS keys for this device are shown below. Use them to register your device with IoT Central. Click to learn more.

Primary Key
+sfrUcn5kUhw2jtedCtTAAhVjO7mV+oCl2voZWMxOGQ=

Secondary Key
1gazGmXRExmwKokDFwzvlMecyq1fv7fTZe3whum5l4w=

take note of the **Scope ID**, **Device ID**, and **Primary Key**.

31. On your local computer where you have this repository,under the Function App directory, there is a folder named **intermediateProcessing** that contains code for the function app that processes data sent from the IoT Hub. Open this folder and change the below values with what you have noted down:

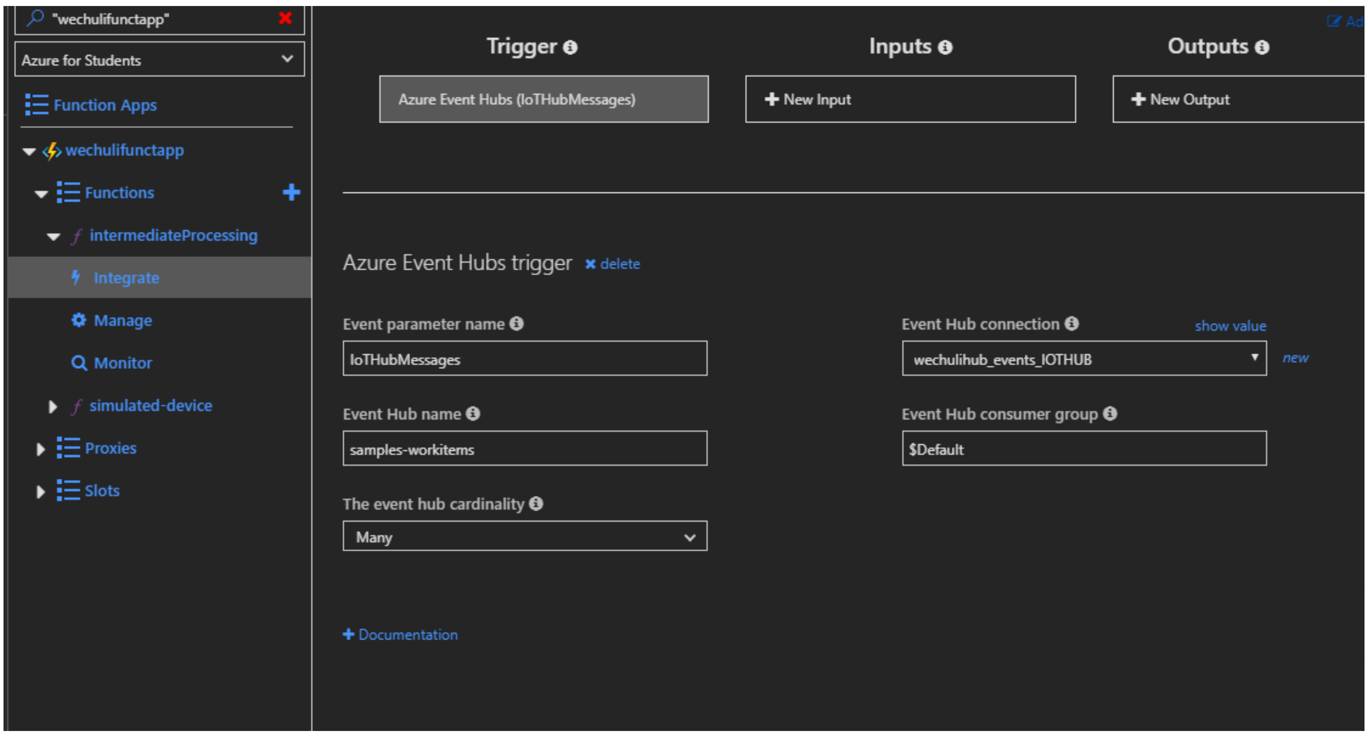
```
const idScope = "{Scope ID}";
const registrationId = "{Device ID}";
const symmetricKey = "{Primary Key}";
```

Remember to save the file.

This function takes data from the Azure IoT Hub, calculates some additional values (chanceOfRain, temprature in Fahrenheit, timestamp, temprature status), before sending the processed data for visualization in the IoT Central Application and Power BI (you will configure this later). All the data processed is stored in Azure Cosmos DB for data persistence.

32. The process of configuring the `intermediateProcessing` function is almost identical to the `simulated-device` function that was configured earlier. Upload the `index.js` to the `intermediateProcessing` function in the cloud. At this point, your application should **NOT** be fully functional because Azure CosmosDB has not yet been configured.

33. Click on the **intermediateProcessing** function in the portal, then choose **Integrate**



Azure Event Hubs trigger ✖ delete

Event parameter name: IoTHubMessages

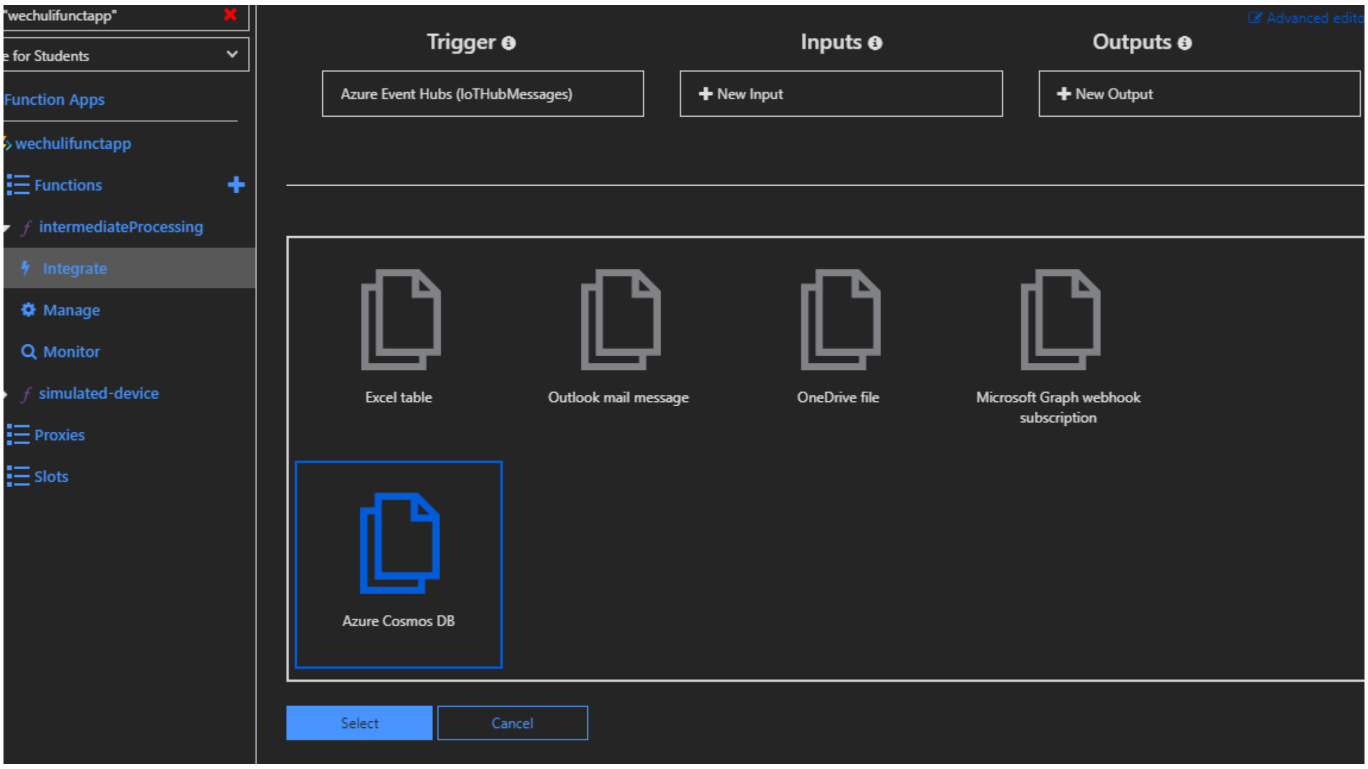
Event Hub connection: wechulihub_events_IOTHUB (show value, new)

Event Hub name: samples-workitems

Event Hub consumer group: $Default

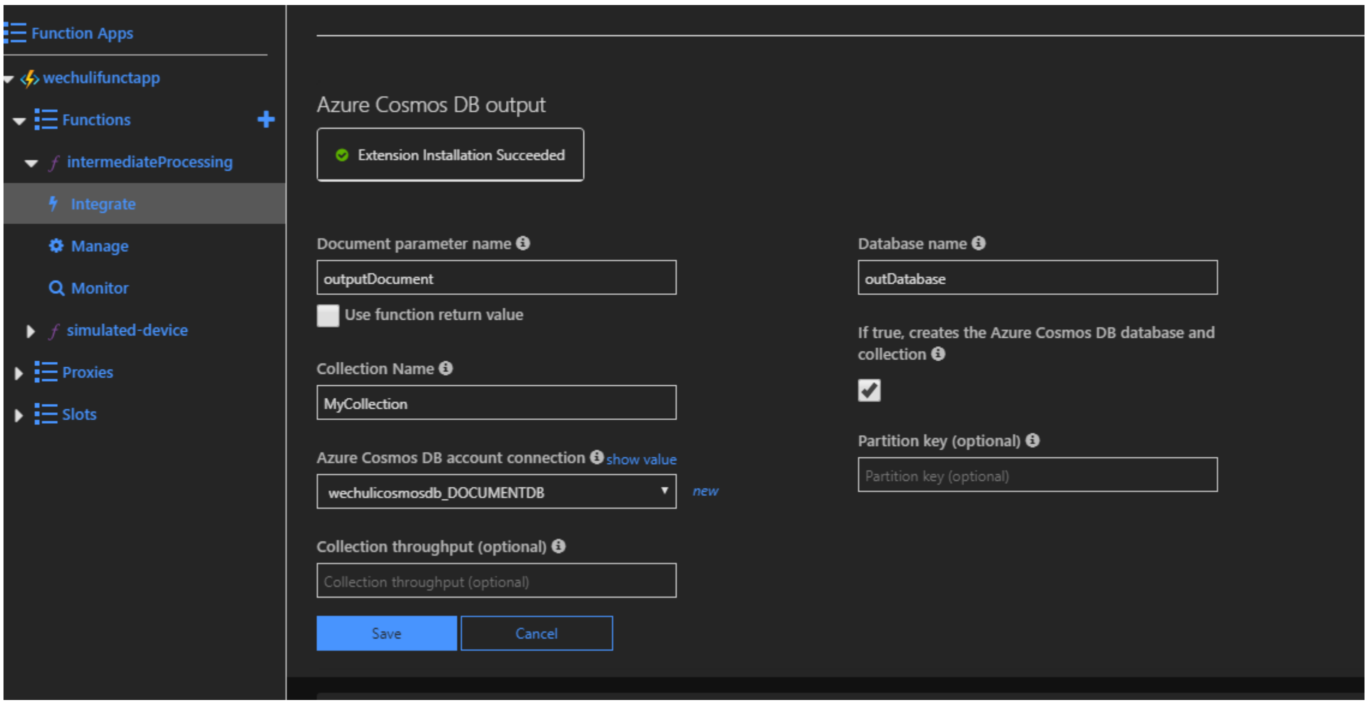
The event hub cardinality: Many

Under **Outputs**, Click the **+** to add a new Output. Choose **Azure Cosmos DB** as your output.



34. You will need to install the CosmosDB extension. When that is done (if it hangs for too long just proceed), confirm the details of the CosmosDB accounts. Select the **If true, creates the Azure Cosmos DB database and collection**.

35. Under **Azure Cosmos DB account connection**, click on ***new***. This opens a popup, which allows you to choose the appropriate Azure Cosmos DB account (the one you provisioined on this resource group). Please note that if you have other Azure Cosmos DB accounts, they will also appear (and you could as well store data in them), so choose the appropriate account. Leave all the other settings as they are. Click **Save**.



Azure Cosmos DB output

Extension Installation Succeeded

Document parameter name: outputDocument

Use function return value

Collection Name: MyCollection

Azure Cosmos DB account connection show value: wechulicosmosdb_DOCUMENTDB new

Collection throughput (optional): Collection throughput (optional)

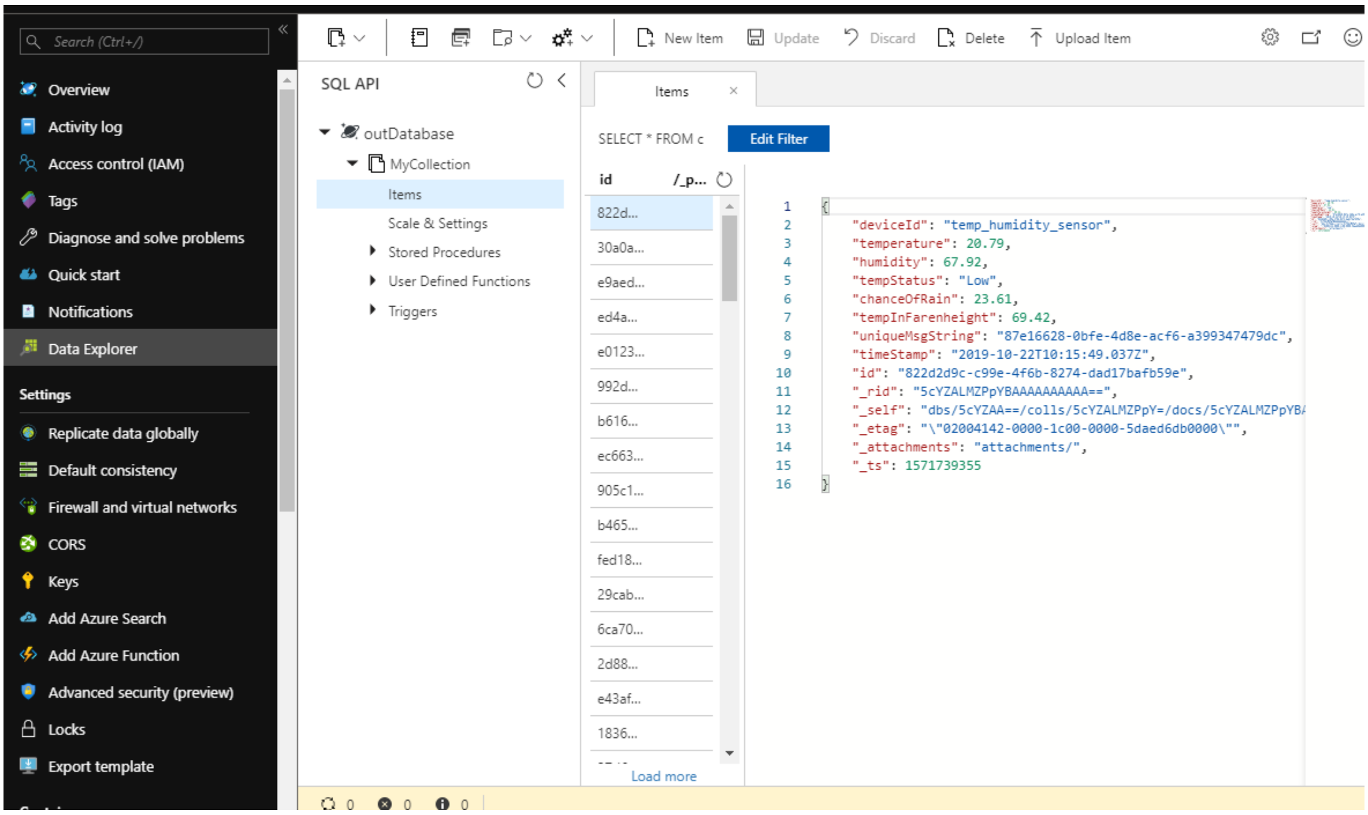
Database name: outDatabase

If true, creates the Azure Cosmos DB database and collection ✓

Partition key (optional): Partition key (optional)

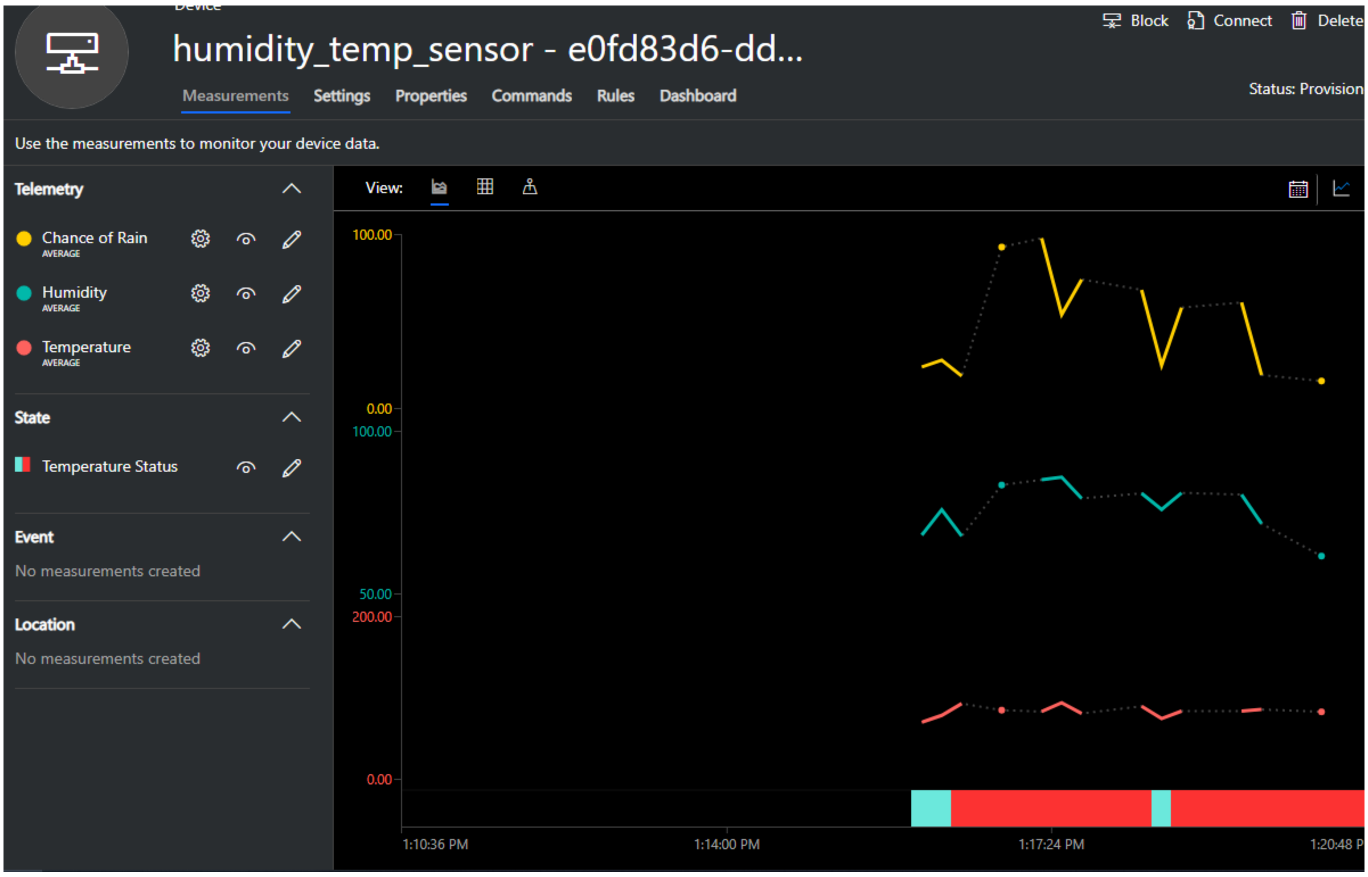
36. You may notice there is a noticable delay before you start seeing output on the logs. (When testing, the logs sometimes hang for an unusual amount of time, if you run into this problem, try restarting the Function App).

37. Confirm that data is being stored in the Azure CosmosDB database. Go to your Resource Group, click on the Azure CosmosDB Account. On the overview pane, near the top there is a **Data Explorer** link that enables you to view your data. You should see a database named `outDatabase`, and a collection called `MyCollection`. Click on `MyCollection` then click on `Items`. You should see a collection of documents that represent the telemetry your device is producing (actually, this data is coming from the Azure function that processes the telemetry). Click on any one of them, you will see a document containing a number of parameters (some are generated automatically, but some were calculated from the `intermediateProcessing` function).

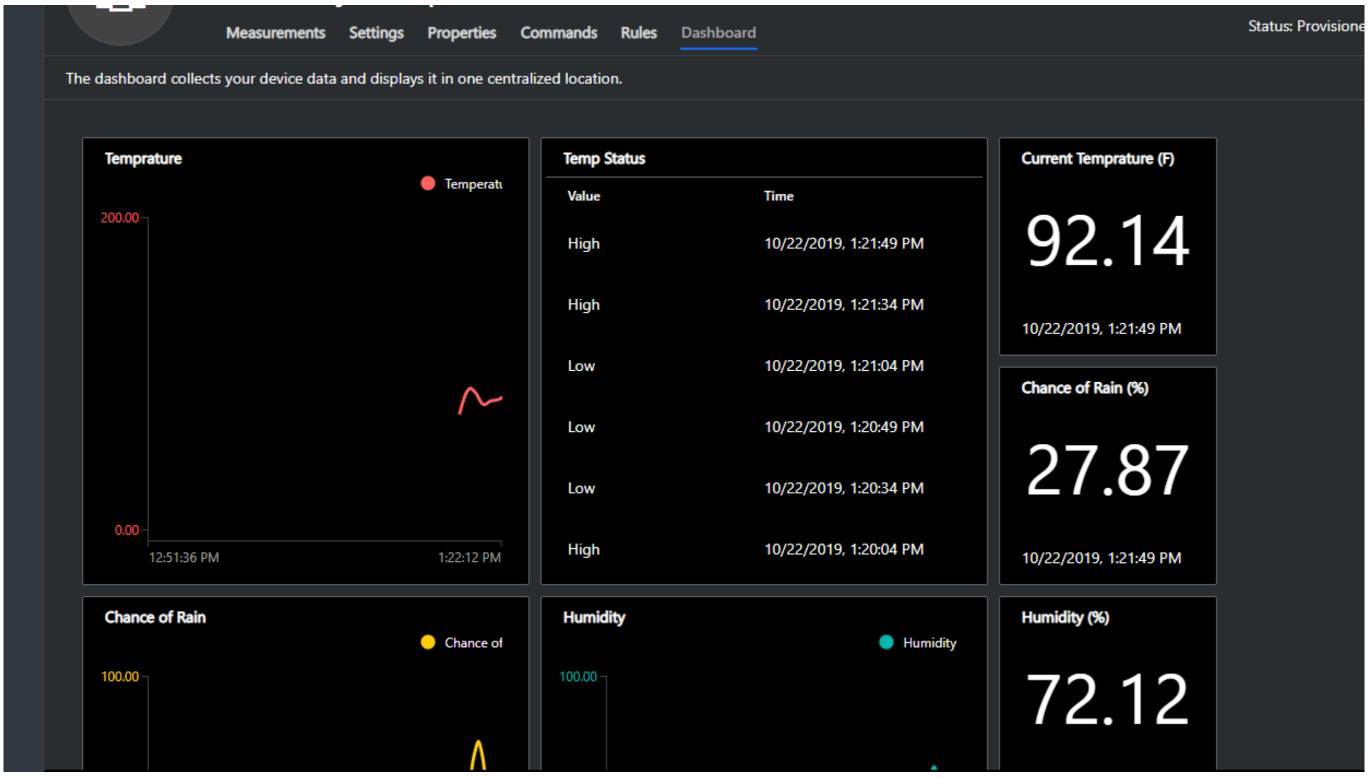


```
{
    "deviceId": "temp_humidity_sensor",
    "temperature": 20.79,
    "humidity": 67.92,
    "tempStatus": "Low",
    "chanceOfRain": 23.61,
    "tempInFarenheight": 69.42,
    "uniqueMsgString": "87e16628-0bfe-4d8e-acf6-a399347479dc",
    "timeStamp": "2019-10-22T10:15:49.037Z",
    "id": "822d2d9c-c99e-4f6b-8274-dad17bafb59e",
    "_rid": "5cYZALMZPpYBAAAAAAAAAA==",
    "_self": "dbs/5cYZAA==/colls/5cYZALMZPpY=/docs/5cYZALMZPpYB
    "_etag": "\"02004142-0000-1c00-0000-5daed6db0000\"",
    "_attachments": "attachments/",
    "_ts": 1571739355
}
```

38. Switch over to your IoT Central Application and locate the real device you registered. On the Device page, you should see data trickling in.



Click on the Dashboard page of that device (still on the Azure IoT Central app), you should see a consolidated visual graphing of the data.



The dashboard collects your device data and displays it in one centralized location.

Temp Status

Value	Time
High	10/22/2019, 1:21:49 PM
High	10/22/2019, 1:21:34 PM
Low	10/22/2019, 1:21:04 PM
Low	10/22/2019, 1:20:49 PM
Low	10/22/2019, 1:20:34 PM
High	10/22/2019, 1:20:04 PM

Current Temprature (F)
92.14
10/22/2019, 1:21:49 PM

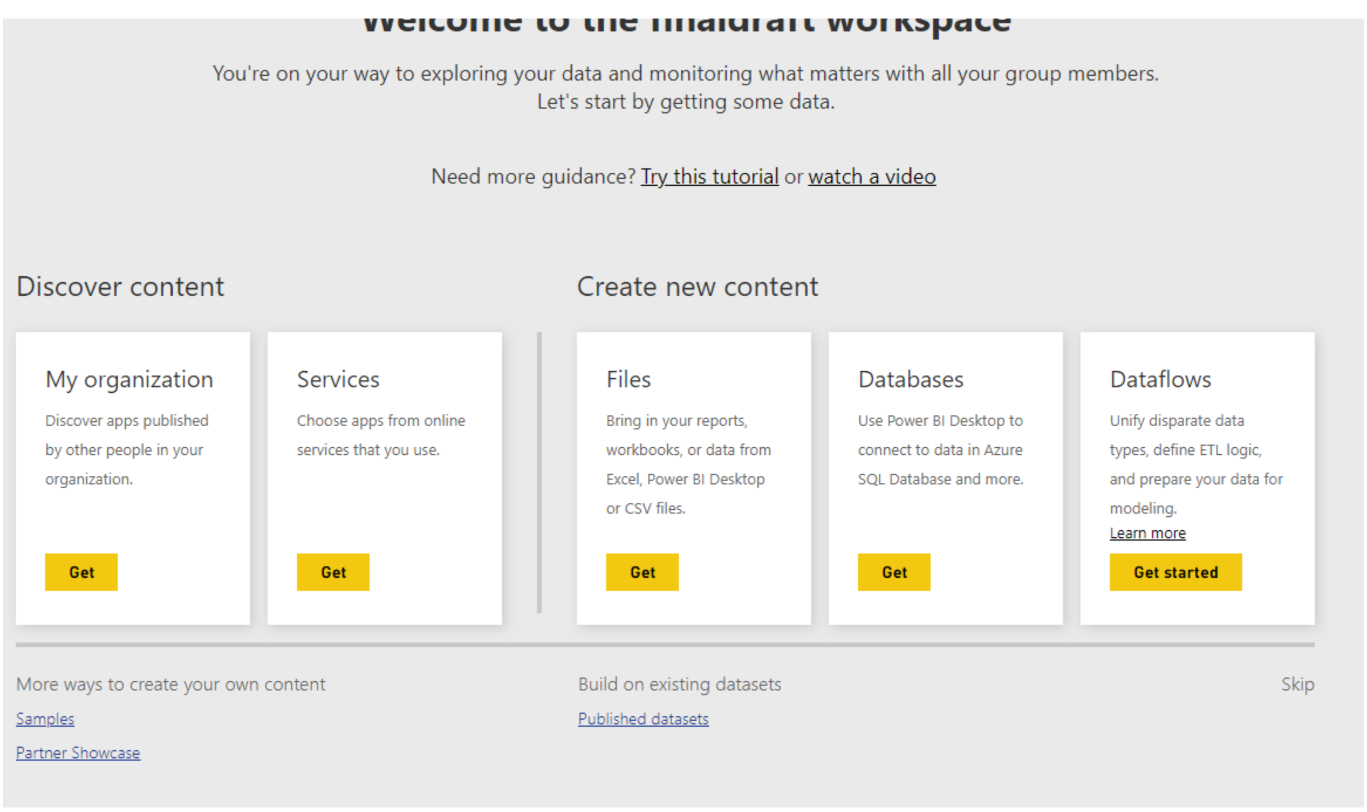
Chance of Rain (%)
27.87
10/22/2019, 1:21:49 PM

Humidity (%)
72.12

Power BI Visualization (Optional)

Power BI provides a convinient way to visualize your data and share insights across your organization. Follow the following steps to attach Power BI visuals to your IoT Data.

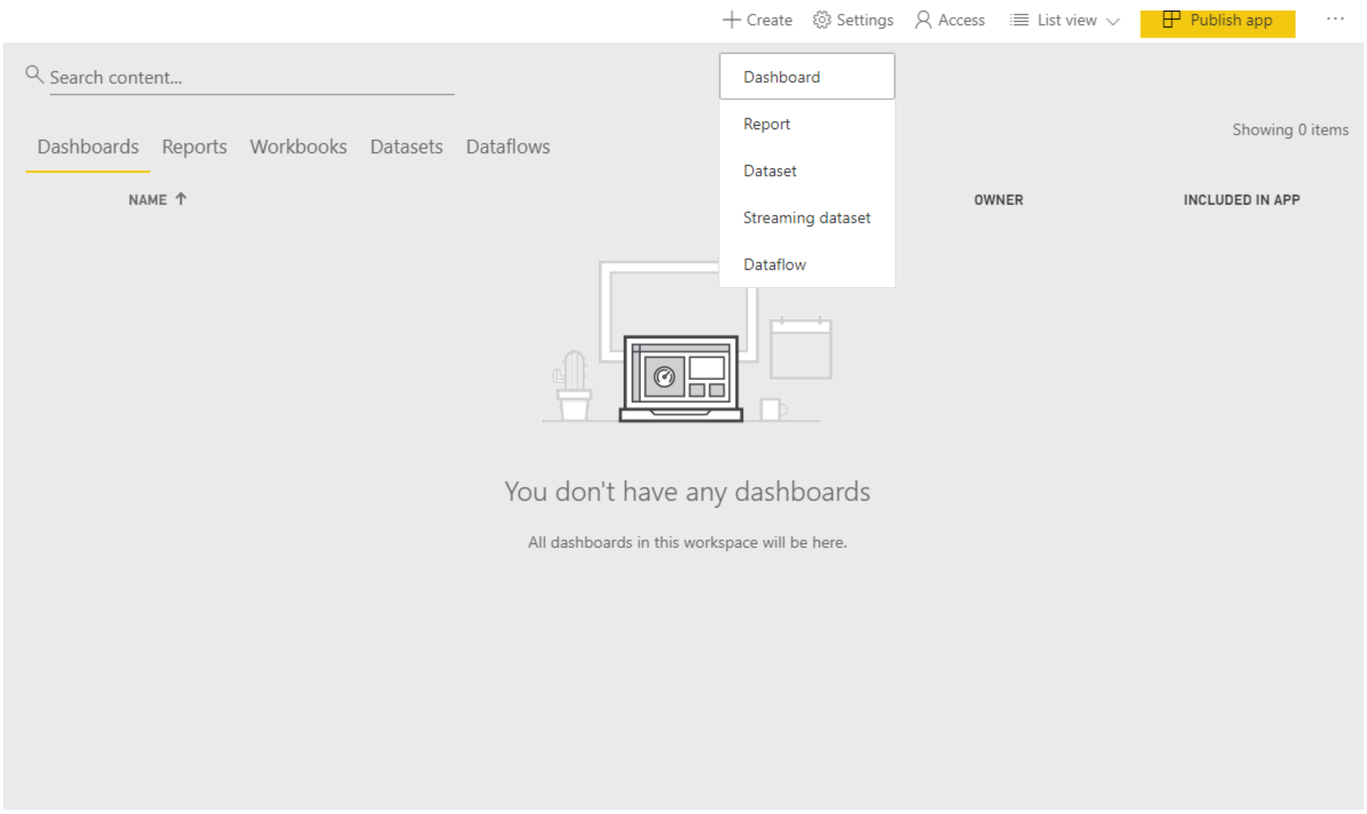
- The first thing you need to do is to sign up for a Pro PowerBI account (you can get a free 60 days trial). Follow this link https://signup.microsoft.com/signup?sku=a403ebcc-fae0-4ca2-8c8c-7a907fd6c235 to do that. Please note Power BI only accepts business addresses, so gmail, yahoo, outlook etc won't work.
- Once you've logged in to Power BI click on **Workspaces**, then **Create a workspace**. You might get a popup asking you to sign up for a free trial, accept it.
- Give the workspace a name and an optional description.Click **Save**. A workspace is basically a container for your dashboards, reports, workbooks and datasets. The first page you should see is the one below.



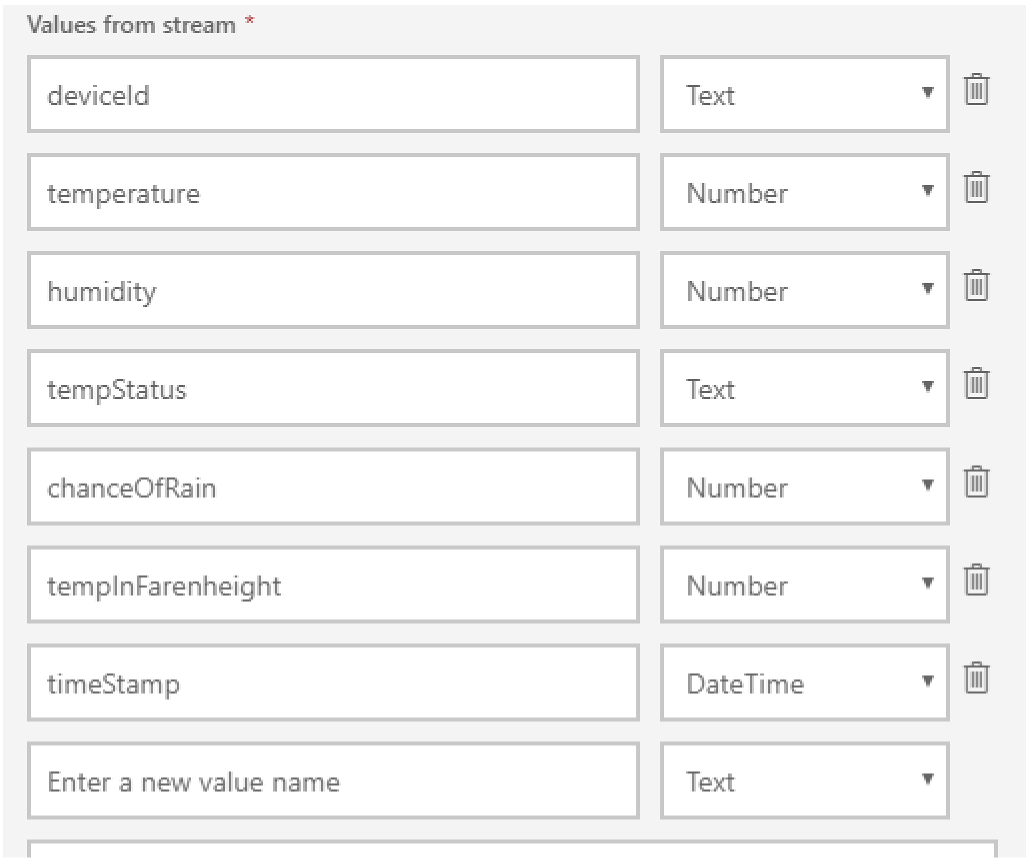
You're on your way to exploring your data and monitoring what matters with all your group members.
Let's start by getting some data.
Need more guidance? Try this tutorial or watch a video
Discover content
Create new content
My organization
Discover apps published by other people in your organization.
Get
Services
Choose apps from online services that you use.
Get
Files
Bring in your reports, workbooks, or data from Excel, Power BI Desktop or CSV files.
Get
Databases
Use Power BI Desktop to connect to data in Azure SQL Database and more.
Get
Dataflows
Unify disparate data types, define ETL logic, and prepare your data for modeling.
Learn more
Get started
More ways to create your own content
Samples
Partner Showcase
Build on existing datasets
Published datasets
Skip

At the bottom right corner, there is a **Skip** link, click on that.

- On the Workspace page click on **Create** at the top, then choose **Streaming dataset**.



- Select **API**, choose *Next*. Give an name to the Dataset and input each value sent from the `intermediateProcessing` function as shown below, make sure you're spelling matches the one show below (this is how the function shapes the data)


Values from stream *

deviceId	Text
temperature	Number
humidity	Number
tempStatus	Text
chanceOfRain	Number
tempInFarenheight	Number
timeStamp	DateTime
Enter a new value name	Text

- Make sure you put **Historic data analysis** on otherwise you can't build reports with the data. Your data should have the shape as shown below:

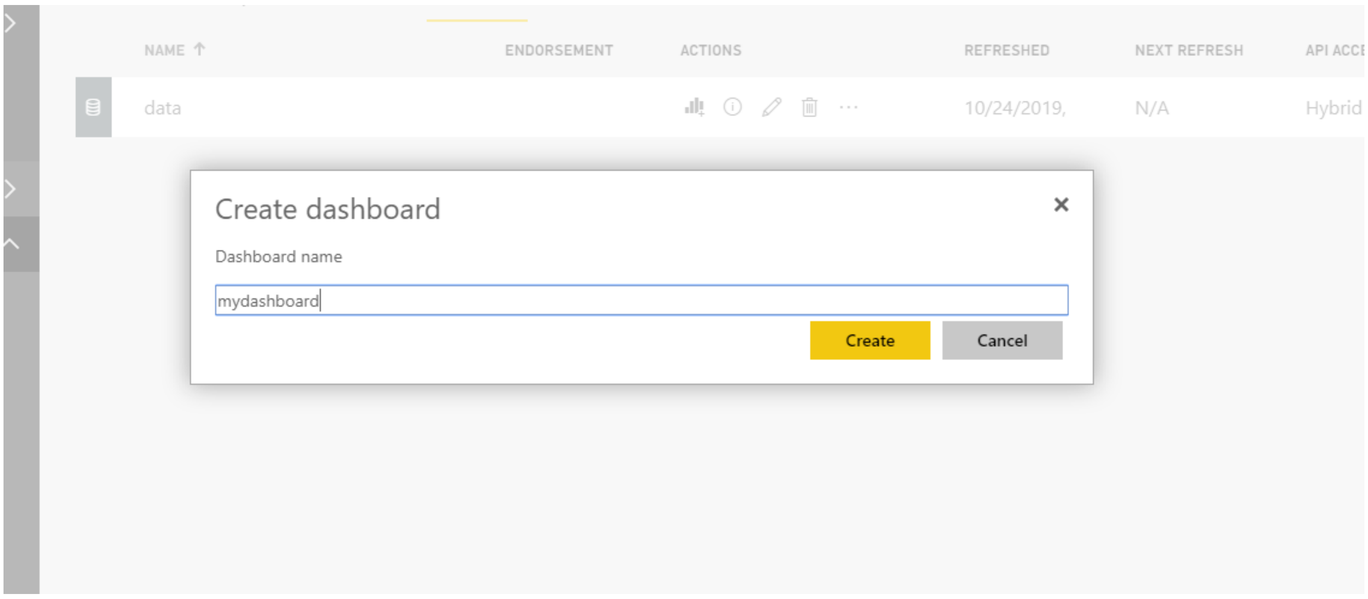
```
[
  {
    "deviceId": "AAAAA555555",
    "temperature": 98.6,
    "humidity": 98.6,
    "tempStatus": "AAAAA555555",
    "chanceOfRain": 98.6,
    "tempInFarenheight": 98.6,
    "timeStamp": "2019-10-24T16:57:21.775Z"
  }
]
```

- Click **Create** and copy the Push URL , click **Done**.
- Head back to the `intermediateProcessing` function on the Azure Portal and open the `index.js` file right in the portal. Locate the part of the code that sends data to Power BI

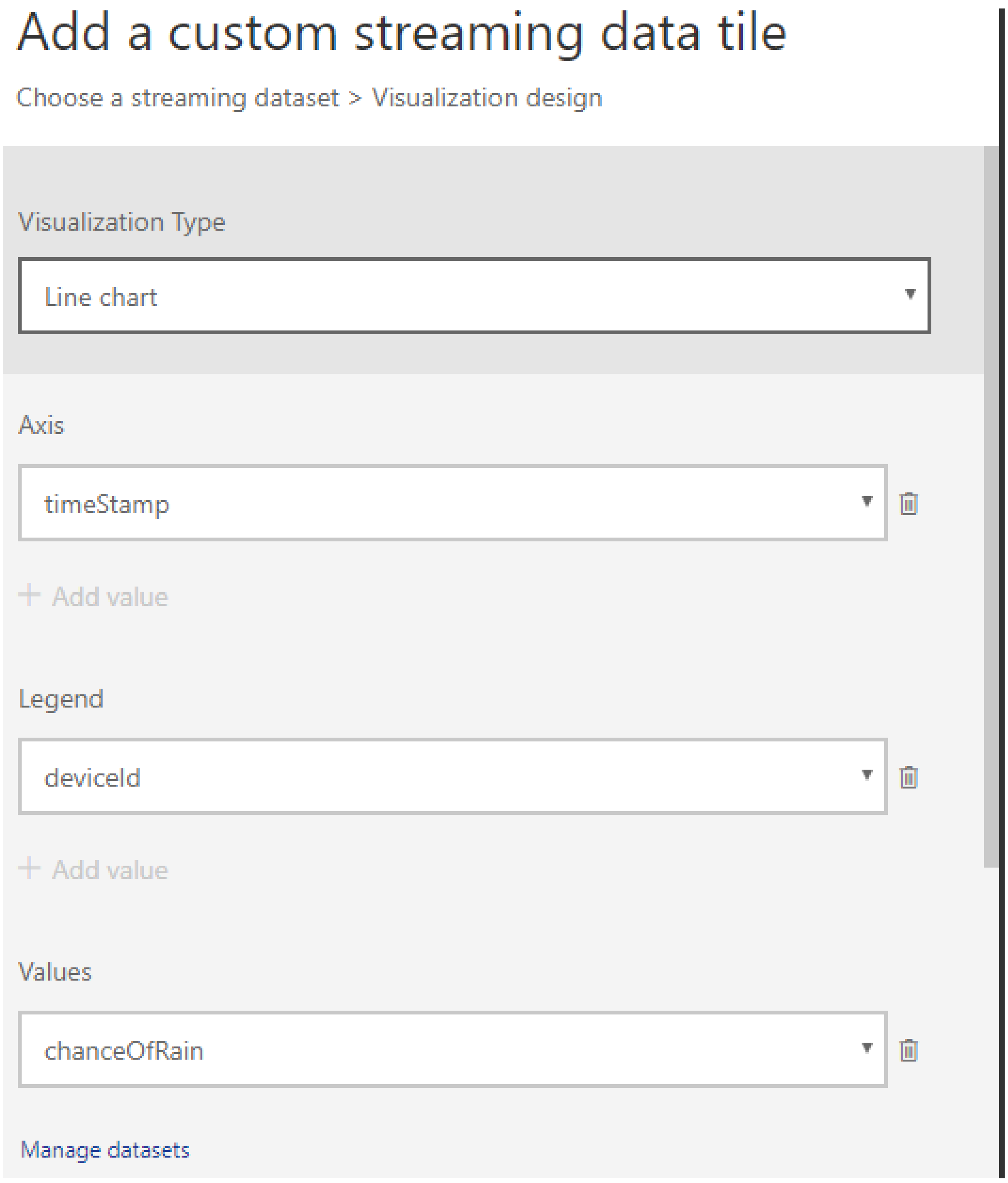
```
axios
    .post("{your power bi push url}", output)
```

Replace `{your power bi push url}` with the Push URL you copied. Save the function. After a little while, it should start running again and data should be streaming to your Power BI service account.

- Back on Power BI, at the top of your screen, select **Create**, then choose dashboard and give the dashboard a name.



- You will be directed to the dashboard, choose **Add tile** at the top them select **Custom Streaming Data** then click on **Next**. Under Your DataSets, select the live streaming dataset you set up then click **Next**. Under `Visualization Type`, choose `Line chart`, choose the Axis to be **timeStamp**, choose the Legend to be **deviceId** and the Values to be **chance of Rain**. Finally, put the Time Window to dispay to 10 minutes (you might want to change this to suite your needs.). Click **Next**. Give the visualization a meaningful name then click **Apply**.



Add a custom streaming data tile

Choose a streaming dataset > Visualization design

Visualization Type: Line chart

Axis: timeStamp

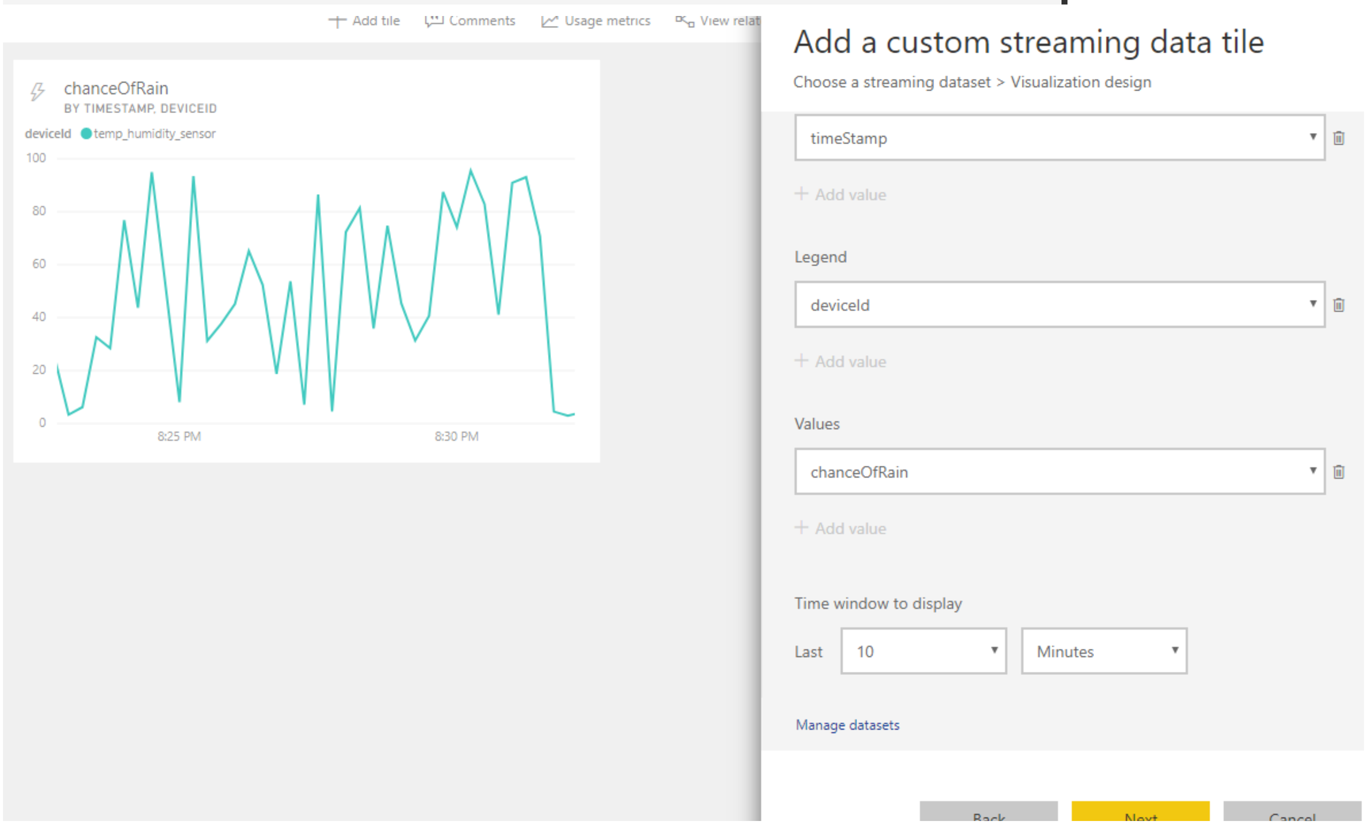
+ Add value

Legend: deviceId

+ Add value

Values: chanceOfRain

Manage datasets



Add a custom streaming data tile

Choose a streaming dataset > Visualization design

timeStamp

+ Add value

Legend: deviceId

+ Add value

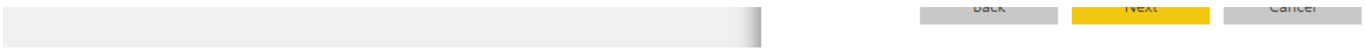
Values: chanceOfRain

+ Add value

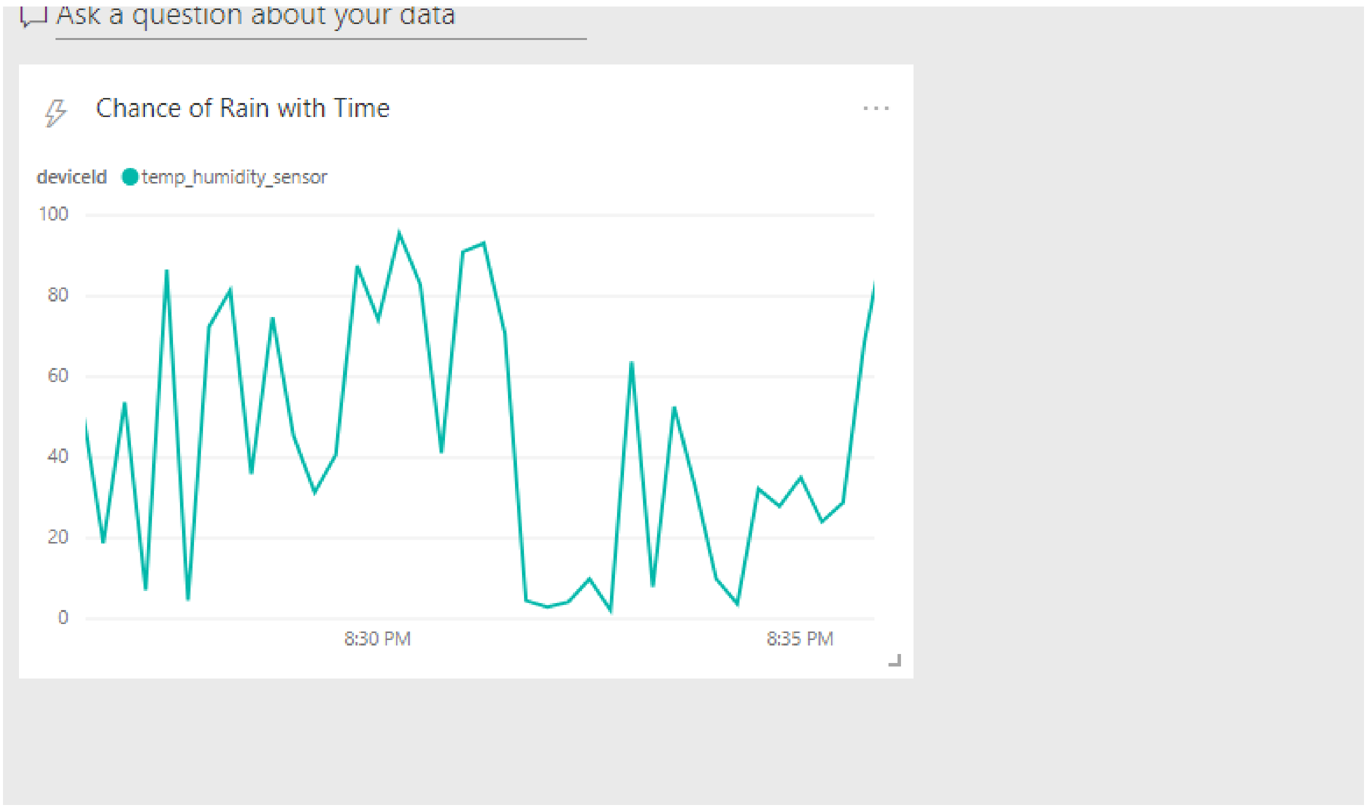
Time window to display

Last 10 Minutes

Manage datasets



- Your dashboard should now have one tile showing realtime readings of the chanceOfRain parameter.


Ask a question about your data
Chance of Rain with Time
deviceId ● temp_humidity_sensor
100
80
60
40
20
0
8:30 PM
8:35 PM

- You can create more visualizations for your streaming data. And you could additionally create reports by clicking on the dataset itself. Reports provide historical analysis for your data, since most useful insights cannot be deduced from plotting only a small timeframe of the streaming data.

Comparison of PowerBI and Azure IoT Central For Visualization

You have used both Power BI and Azure IoT Central to visualize your data but there are some key and important difference between the two techniques. In fact, you could use both of them since they are meant for different use cases.

Power BI

This is a business intelligence too (hence then name PowerBI) .Power Bi would be suitable as a visualization and analytics tool when data being consumed is needed within the organization.

Pros

- You can easily drag and drop components to create visualizations that you want
- You can build custom reports and extract insights from the reports and dashboard.
- Large array of visualization options and widgets.

Cons

- Not suitable if you intend to share visualizations with parties outside your organization.

- Every user that the report or dashboard is shared with must have a Power BI Pro account, otherwise they can't access the content. A Pro Licence is $9.99/month per user

Azure IoT Central

Azure Central is a SaaS IoT solution, so it includes more than just device data visualization. The system you built only used it for visualization but it can be used for much more.It is suitable for situations where you want a quick IoT solution without having to worry about setting up infrastructure and writing custom applications.This service is for straightforward solutions that don't require deep service customization.

Pros

- Easy to set up
- prebuilt visualization that automatically refresh and update with a template you define
- Ability to simulate a device
- Easy to share access to the application to anyone with a Microsoft Email Account (Outlook.com)
- Can build alerts for devices to be notified when something goes wrong.

Cons

- There is no way for you to preprocess the device data before it reaches the IoT Central App since you don' t have access to the underlying infrastructure
- You cannot create custom visualizations and reports
- Data is automatically cleared after 30 days so you have to build integration to store this data yourself.
- Free for the first 5 devices but $2/month per device thereafter.

Clean Up The Resources

To delete all of the resources at once run the following command, be sure to change the `--name` parameter to whatever you called your resource group.

```
az group delete --name ExampleGroup
```